3D-Audio with CLAM and Blender's Game Engine

Natanael Olaiz, Pau Arumí, Toni Mateos and **David Garcia.**
Barcelona Media Centre d'Innovació
Av. Diagonal, 177, planta 9,
08018 Barcelona, Spain.
{natanael.olaiz, pau.arumi, toni.mateos, david.garcia}@barcelonamedia.org

Abstract

Blender can be used as a 3D scene modeler, editor, animator, renderer and Game Engine. This paper describes how it can be linked to a 3D sound platform working within the CLAM framework, making special emphasis on a specific application: the recently launched *Yo Frankie!* open content game for the Blender Game Engine. The game was hacked to interact with CLAM, implementing spatial scene descriptors transmited over the Open Sound Control protocol, and allowing to experiment with many different spatialization and acoustic simulation algorithms. Further new applications of this Blender-CLAM integration are also discussed.

Keywords

3D-Audio, CLAM, Blender, Game Engine, Virtual Reality

1 Introduction

The Blender project [1] has done an impressive effort to produce demonstrators of their technologies, such as the *Elephants Dream* [2] and *Big Buck Bunny* [3] films for 3D movie production capabilities, and *Yo Frankie!* [4] for its Game Engine (GE). The needs introduced by those realizations have been a main driving force to enhance the platform itself.

On the other hand, the CLAM framework [5] has recently incorporated a set of real-time 3D-audio algorithms, including room acoustics simulation, and Ambisonics coding, manipulation, and decoding. This paper discusses an attempt to exploit some of the nice Blender features to create an experimental platform that links Blender, specifically its Game Engine (GE) for the graphics and interaction part, with CLAM for the spatialized audio.

The linking of Blender and CLAM has been achieved via the Open Sound Control (OSC [6]) protocol and a customized version of the Spatial Sound Description Interchange Format (SpatDIF [7]). This architecture still renders both systems rather decoupled and independent, providing great flexibility. In particular, it maintains the simplicity of independently changing both the geometrical elements that define the scenes and the audio algorithms in the CLAM audio dataflow-based processing networks.

Compared to other existing 3D-audio systems that integrate in gaming engines, CLAM offers a rich variety of possibilities regarding exhibition systems and room acoustics simulation. For example, the game sound can be exhibited in multi-loudspeaker setups, not only surround 5.1, but virtually any setup, including those using speakers at different heights. It can also output binaural audio for earphones, based on HRTF's; CLAM uses different binaural techniques and allows switching between different public HRTF's databases.

Moreover, CLAM offers the possibility of performing room acoustics simulation using ray-tracing techniques, making it possible to recreate very realistic 3D reverberations in real-time. This allows the sound reverberation to dynamically change whenever the sound sources or listener change their positions.

The division of responsibilities between the two platforms goes as follows: Blender manages the 3D scene and sends objects positions, orientations and actions to a CLAM processing network. Then, CLAM plays the appropriate audio streams, optionally performing room acoustics simulation, and finally renders the audio to the desired exhibition system, either a multi-speaker setup or headphones. Of course, when room acoustics simulation is enabled, CLAM requires a copy of the 3D model used in Blender to apply its algorithms.

Though the system that we present today still comprises two rather decoupled applications that communicate though OSC, in the future is likely that CLAM will be integrated into Blender as a library, similar to (or extending) OpenAL [8]. This would be beneficial in terms

of run-time efficiency, ease of use and software distribution.

The paper is organised as follows. We start by describing the relevant parts of the Blender engine (section 2) and the CLAM 3D-audio platform (section 3). In section 4 we describe the communication between them. Section 5 discusses other uses that this integration provides. In section 6 we present our conclusions and a description of future work.

2 Blender

Besides the use as a standard 3D application, the Blender functionalities can be easily extended using Python scripting. Blender includes two main Python APIs: the (*bpy*) modules for the main editor and preview user interface; and the GameLogic modules for the Game Engine, which allow user interaction and physics simulations using a simple logic blocks design. In what follows we concentrate on the latter.

In the Game Engine, the events are generated by *sensors*, and sent to the *controllers*, which decide what action should be performed and interact with the scene either through *actuators* or Python scripts (see fig. 1). Note that the sensors are not only attached to keyboard or mouse events, but to higher level concepts like proximity to, or collision with, other objects.

Although an exhaustive explanation of the *Yo Frankie!* features is outside the scope of this paper, suffice it to say that it comprises the aforementioned three main components of the GE logic. In our implementation, some of the sensing events trigger custom Python scripts within the *controllers* block that communicate with CLAM.

Figure 1: Blender Game Engine logic blocks.

We use Python scripts to communicate a number of different data to CLAM. On the one hand, a Python plugin allows to first assign acoustic properties (like impedance or diffusion coefficients) to the materials present in the geometry, and then export the scene in a format usable by the room acoustics simulator. On the other hand, Python scripts also act as controllers that transmit to the CLAM audio platform the necessary information about the sound sources and listener, mostly 3D positions and orientations, and source directivity patterns. As mentioned above, this communication is based on the SpatDIF protocol over Open Sound Control.

Whereas non-animated sound sources (e.g. trees) have their positions and orientation angles sent over OSC when the user starts the GE, those that are animated and interactive (including the listener) send data constantly. Usually the sound sources send a control message of loop sample play, and trigger specific samples when some special actions are triggered (a kick makes *Momo*, the monkey, scream, for instance).

Although the original game has its own sounds, and uses OpenAL for a more or less complex processing, the objective was to have a development platform to test and explore different new algorithms (e.g. ray-tracing room acoustics); this is more easily accomplished within CLAM than within OpenAL, and without hardware requirements.

3 3D-Audio in CLAM

3.1 CLAM

CLAM is a C++ framework for audio processing, with GPL license, created to support audio and music research and rapid application development [5; 9]. Its processing core is based on the dataflow processing model, where a user defines a system in terms of a graph of processing objects —or "networks" in CLAM's nomenclature. One of the particularities of CLAM is that it supports different token types, such as audio buffers and spectrums, and different ports-consumption rates (e.g. a spectral domain processing may run less times per time unit than an audio domain processing). The scheduling of CLAM network's processings can be computed offline, before run-time, using the Time-Triggered Synchronous Dataflow model [10].

CLAM supports both real-time and offline operation. The first mode allows interacting with other systems via Jack [11] or embedding the network in an audio plugin, of any architecture, such as LADSPA [12]. The second mode allows defining complex work-flows by using scripting languages, such as Python, and compute CPU intensive tasks that can not run in real-time.

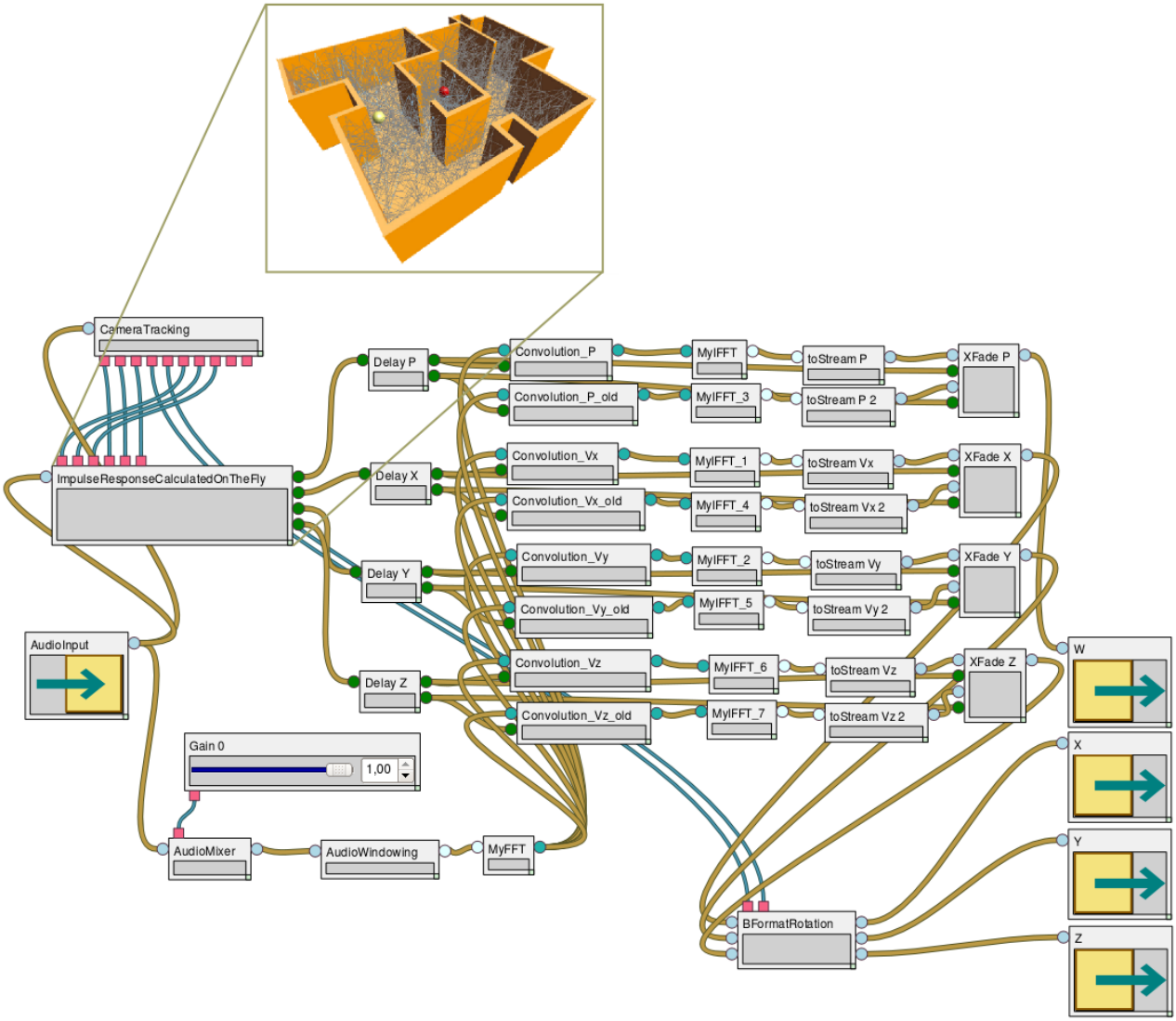

Figure 3: CLAM network that renders the audio in B-Format.

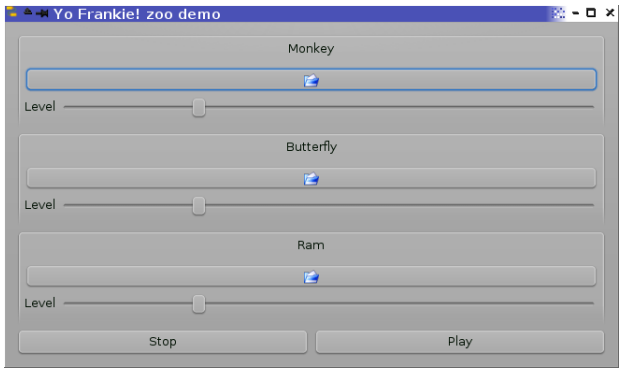

Figure 2: OSC receivers, sampler and spatialization, running with CLAM Prototyper Qt interface.

3.2 The 3D-Audio Engine

The following paragraphs describe the main CLAM networks used in our 3D-audio engine. The network depicted in figure 3 shows the processing core of the system. It produces 3D-audio from an input audio stream, plus the position of a source and a listener in a given 3D geometry —which can also be an empty geometry. If the room-simulation mode is enabled, the output audio contains reverberated sound with directional information for each sound reflection, therefore producing a sensation of being immersed in a virtual environment to the user.

The format of the output is Ambisonics of a given specified order [13; 14]. From the Ambisonics format, it is possible to decode the audio to fit diverse exhibition setups such as HRTF-based audio through headphones, standard surround 5.1 or 7.1, or other non-standard loudspeakers setups. Figure 5 shows a CLAM

network that decodes first order Ambisonics (B-Format) to surround 5.0, whereas the network in figure 6 decodes B-Format to binaural.

Let us describe in more detail the main audio rendering network, depicted in figure 3. The audio scene to be rendered is animated by a processing which produces controls of source/listener positions and angles. This processing can be either an OSC receiver or a file-based sequencer. The picture illustrates this second case, where the *CameraTracking* processing sequences controls from a file (for instance, exported from Blender) and uses an audio input for synchronization purposes.

The actual audio rendering process is done in two stages. The first stage consists on the computation of the acoustic impulse-response (IR) in Ambisonics format for a virtual room at the given source/listener positions. This process takes place in the *ImpulseResponseCalculatedOnTheFly* processing which outputs the IRs. Since IRs are typically larger than an audio frame they are encoded as a list of FFT frames.

The second stage consists on convolving the computed IRs, using the overlap-and-add convolution algorithm, which is depicted in figure 4 and explained in [15]. This process is implemented in the *Convolution* processing which takes two inputs: a FFT frame of the incoming audio stream and the aforementioned IR.

The IRs calculation uses acoustic ray-tracing algorithms [1] , which take into account the characteristics of the materials, such as impedance and diffusion. The IR calculation is only triggered by the movement of the source or listener, with a configurable resolution.

First informal real-time tests have been carried out successfully using simplified scenarios: few sources (3), simple geometries (cube), and few rays (200) and rebounds (70). We are still in the process of achieving a physically consistent reverberation by establishing the convergence of our ray-tracer (i.e. making sure that we compute enough rays and enough rebounds to converge to a fixed RT60). We will include a discussion on this in further papers. Another future line is optimize the algorithm for real-time by reusing or modeling reverberation tails and only compute the early reflections by ray-tracing.

As the diagram shows, each B-Format component (W, X, Y, Z) of the computed IR is produced in a different port, and then processed in a pipeline. Each branch performs the convolution between the IR and the input audio, and smoothes the transitions between different IRs via cross-fades in the time domain. The need for such cross-fades requires doing two convolutions in each pipe-line.

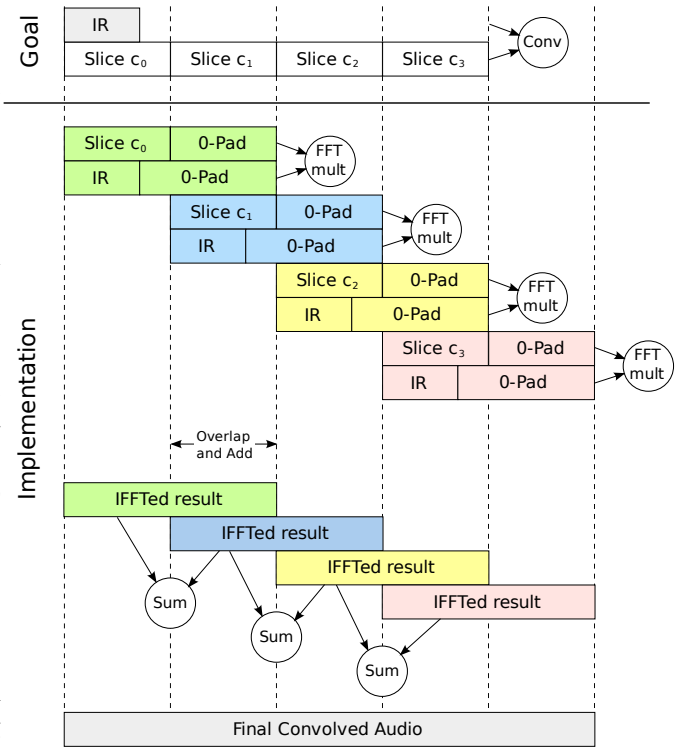

Figure 4: A schematic description of the partitioned convolution processing object.

One last source of complexity: since sources and listener can move freely, cross-fades are not enough. The result of an overlap-and-add convolution involves two IR's which, among other differences, present different delays of arrival in the direct sound and first reflections. When such differences are present, the overlap-and-add will have a discontinuity or clip, which the user notices as an annoying artifact.

This problem has been solved in this case by taking advantage of the two branches that were already needed for cross-fading the IR transition. The key point is to restrict how IRs

[1] At the moment, the ray-tracing implementation for room-acoustics simulation is not open-source.

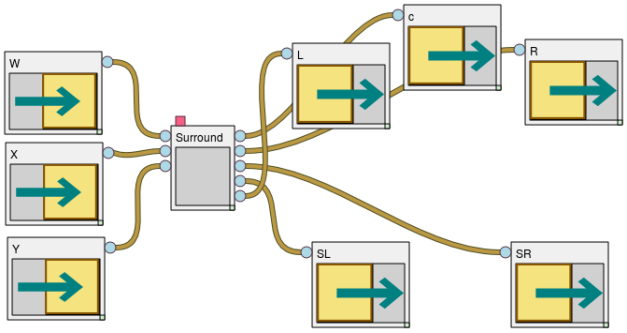

Figure 5: A simplification of the partitioned-convolution algorithm performed by the CLAM's "Convolution" processing.

change, so that only one branch can be clipped at a time. With this restriction the problem can be solved by means of the *XFade* and *Delay* processings. The *Delay* processing produces two IR outputs: the first is just a copy of the received IR and the second is a copy of the IR received in the previous execution. To ensure that, at least, one overlap-and-add will be clip-free this processing will "hold" the same output when a received IR object only lasts one frame. The *XFade* reads the IR objects identifiers — and hence, its 4 input ports— and detects when and which branch is carrying a clipped frame, to decide which branch to select or to perform a cross-fade between the two.

In the last step, the listener's orientation is used to rotate the B-Format accordingly. The rotated output is then ready to be streamed to one or more decoders for exhibition systems.

4 Communication between Blender and CLAM via SpatDIF

For the OSC interaction, we used the SpatDIF protocol, which aims at establishing an open standard interchange format for spatial sound description [16], illustrated in figure 7. The definition of SpatDIF is still work in progress, given the fact that at the moment the major published information have been only its goals and a few use examples [7]. The present implementation for the GameEngine-CLAM OSC communication has made use of a subset of the SpatDIF standard, extended to cope with the needs mentioned in this paper. We plan to suggest that some of these extensions be added to SpatDIF, and expect to implement full use of this standard when finished, including its non-real time implementation.

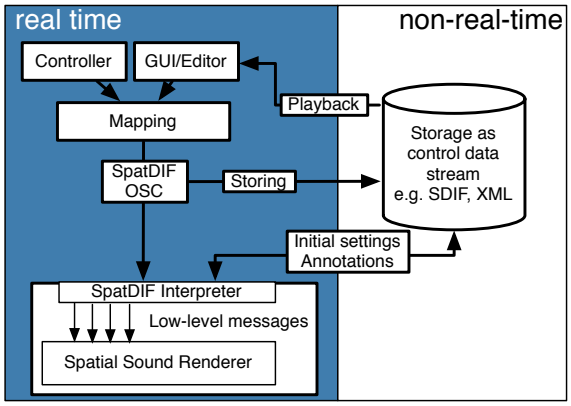

Figure 7: SpatDIF work blocks diagram. Copied from *Proposing SpatDIF - The Spatial Sound Description Interchange Format.* [16] with author permission

For the sake of illustration, let us present some examples of SpatDIF messages used:

- `/SpatDIF/source/`*n*`/xyz (x-axis y-axis z-axis ; floats)`
- `/SpatDIF/source/`*n*`/aed (azimuth elevation distance ; floats)`

Both examples describe the position of a sound source *n*. The first uses absolute coordinates on a 3D reference system, whereas the second uses spherical coordinates. Note that, in our implementation, besides the use of an integer, it is also possible to use a string to univocally refer to an object name.

Examples related to source orientation and pattern directivity are :

- `/SpatDIF/source/`*n*`/aer (azimuth elevation roll ; floats)`
- `/SpatDIF/source/`*n*`/emissionPattern (`*e*`, where` *e* `can be either a string or an integer, pointing to one of the source emission pattern directivities of a given table,` *e.g.* `omni or cardioid)`

We have also needed to send setup/triggering messages to control a sampler based sound, working with layers, and allowing any sound source to trigger more than one sample at a given time. Some examples:

- `/SpatDIF/source/`*n*`/sampler/addLayer (name ; string)`
- `/SpatDIF/source/`*n*`/sampler/`*name*`/setBuffer (audio file name ; string, to define the sampler layer`

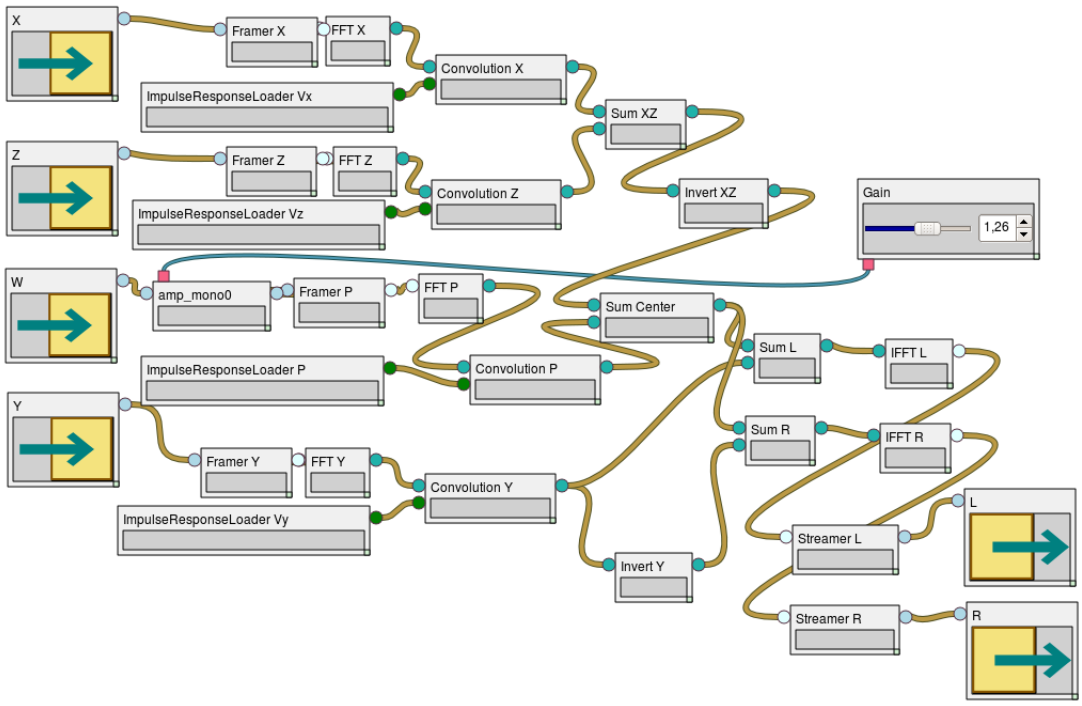

Figure 6: CLAM network that converts 3D-audio in B-Format to 2 channels binaural, to be listened with headphones. The technique is based in simulating how the human head and ears filters the audio frequencies depending on the source direction.

```
used sound file)
```

- `/SpatDIF/source/`*n*`/sampler/`*name*`/setLoop (bool ; sampler layer loop configuration)`

- `/SpatDIF/source/`*n*`/sampler/`*name*`/play (bool ; play/stop control)`

For the purpose of real time preview on Blender (not Game Engine), we have incorporated a frame sync message as follows:

- `/SpatDIF/FrameChanged (number of frame ; integer)`

Another extension was needed to define an initial setup message to add sources, analog to the work of the sampler *addLayer* one:

- `/SpatDIF/source/addSource (name ; string)`

In a future extension, it would be desirable to standardize a format for geometry and acoustic properties of materials suitable for room acoustics calculations. This would enable the communication of such data when the Game Engine starts, thus avoiding the initial offline exportation.

5 Other uses

Given that the Game Engine allows real time user interaction with good quality image OpenGL rendering, and given that the correspondent CLAM Networks can do the counterpart for audio, we think that the following use cases make a fruitful use of this integration, and we are experimenting with them.

5.1 Real-time animation preview

The main user interface of Blender allows the creation, manipulation and previewing of a scene (objects and animations) in real-time and, as in the Game Engine, with an OpenGL rendering quality. Using almost the same aforementioned scripts, it is also possible to send the positions and orientations of objects defined as listeners and sources in real time, thus providing a new powerful 3D sound spatializator editor interface and previewer. In this sense, we can think of the Blender-CLAM integration as a kind of *what you see if what you hear* editor.

5.2 High quality offline rendering

The Blender-CLAM integration offers an audio counterpart of high-quality offline image rendering. As mentioned above, besides sound

sources and listener data, the CLAM spatialization plugin includes exporters of Blender geometries with acoustic parameters associated to the materials present in the scene. Using the CLAM OfflinePlayer, it is possible to generate high quality offline room acoustics simulations, which recompute the corresponding reverb every time a source or the listener moves. The result of the offline rendering, which can be in Ambisonics format with any given order, can then be used as an input on a Digital Audio Workstation (DAW) for further post-production. At the moment, we use Ardour LADSPA plugins developed for that purpose.

For this offline application, we are indeed using an specific CLAM file format which contains all the parameters describing the motion of sources and listener at each video frame, the zoom of the camera, etc. In the mid-term, we plan to use the same SpatDIF format in both real-time and offline applications (fig. 7).

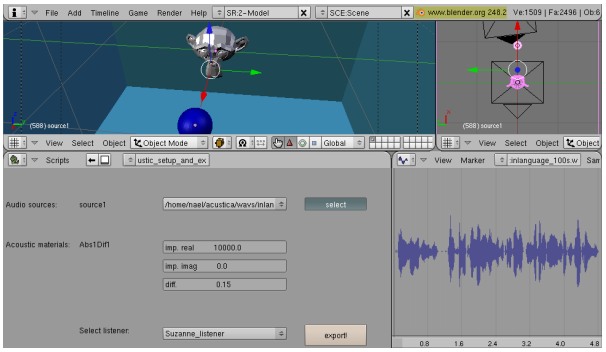

Figure 8: Exportation of Blender scene geometries and animations.

5.3 Head-tracking and motion sensors

One interesting possibility to explore with this platform is the use of motion sensors to interact with Blender. For instance, head-tracking applied to binaural simulations improve the interactive 3D sound immersive experience. Another example is the use of a tracker to define three-dimensional trajectories that can be assigned to objects in the scene with the aim of spatializing the sound associated to them. This can be thought of a sort of 3D extension of the joysticks used in DAWs for panning sounds.

6 Conclusions and future work

It has been shown that combining Blender and CLAM is useful to render 3D-audio by manipulating 3D scenes, and it provides a flexible and

Figure 9: Combining *Yo Frankie!* with head-tracking.

powerful platform for experimenting with different acoustic field rendering techniques and exhibition setups.

An interactive game and several short movies have already been produced using these tools in the Barcelona Media audio lab. The game is a modified version of Blender's *Yo Frankie!*, which communicates the position and sound events of the actors to CLAM using an OSC based protocol. The short movies have been post-produced using Blender for 3D animation and a plugin developed to associate geometric objects with sound sources and finally export this data for offline audio rendering using CLAM.

The 3D-audio for both the game and the short movies has been exhibited in different setups, including binaural, stereo, 5.1, and a custom 15-loudspeakers setup, the latter distributed as follows: six in a central rig, four in a upper rig, four in a lower rig and one speaker on top (see fig. 10). For these initial tests, the audio has been decoded using VBAP and first order ambisonics (B-Format), but other techniques that make a better use of a large number of loudspeakers are currently being implemented, among these, higher order Ambisionics.

Initial informal tests have been performed with people from the audio-visual industry in general and sound design in particular, and the results have been so far encouraging.

The rationale behind the use of a 3D graphics tool, such as Blender, for audio purposes is that most of today's audio-centric tools used by professionals in media post-production, such as Digital Audio Workstations (DAW's), largely

Figure 10: Picture of the arrange of 15 speakers used whithin the development.

lack support for geometric metaphors. It is likely that the increasing interest of the industry for 3D cinema and TV will also push the interest for 3D-audio content, which in turn, will demand for new or modified audio tools. We believe that a simplified set of Blender's features allowing for scene manipulation shall be incorporated into the DAW's core features, allowing the production of 3D-audio content independently of the final exhibition format (such as 5.1, 22.2, binaural or Wave Field Synthesis).

The presented work still has many open lines. First, more encoding and decoding algorithms such as higher order Ambisonics (specially in non-regular setups) should be tested within the real-time game. Second, the way sound samplers are set up in CLAM should be made more flexible. As it stands, the addition of more sound sources (and hence, samples) implies manually modifying a CLAM network. In short, this CLAM network setup will be done automatically by means of the proposed SpatDIF protocol extension, transmitting the scene information from Blender to CLAM. Third, non punctual sound sources should be implemented, for example, a river. In this case the sound should emanate from all points in a line (the river) instead of a single point, possibly incorporating decorrelation algorithms to increase its spatial extend perception [17]. Fourth and last, all the audio functionality could be encapsulated into a library with a well defined interface, maybe using and extending the OpenAL API. This would probably enable enhanced run-time performance, easier reuse and better applications deployment. However, this is work should be done only once the system is mature enough and with stable functionalities.

Acknowledgements

This work is supported by the European Union FP7 project 2020 3D Media (ICT 2007) and by the Google Summer of Code 2008 program. We are specially thankful to all CLAM and Blender developers.

References

[1] Blender Foundation. Blender main website. `http://www.blender.org/`.

[2] Blender Foundation Orange Open Movie Project. Elephants dream open movie main website. `http://orange.blender.org/background`.

[3] Blender Foundation. Big buck bunny open movie main website. `http://www.bigbuckbunny.org/index.php/about/`.

[4] Blender Institute. Yo frankie! main website. `http://www.yofrankie.org/about-apricot`.

[5] X. Amatriain, P Arumi, and Garcia D. A framework for efficient and rapid development of cross-platform audio applications. *ACM Multimedia Systems*, 2007.

[6] UC Berkley Center. Open sound control main website. `http://opensoundcontrol.org/introduction-osc`.

[7] Nils Peters et all. Spatdif main website. `http://spatdif.org/`.

[8] Creative Labs. Openal main website. `http://connect.creativelabs.com/openal/default.aspx`.

[9] CLAM team. Clam framework main website. `http://clam.iua.upf.edu`.

[10] P. Arumi and X. Amatriain. Time-triggered Static Schedulable Dataflows for Multimedia Systems. *Proceedings of Multimedia Computing and Networking*, 2009.

[11] P. Davis, S. Letz, Fober D., and Y. Orlarey. Jack Audio Server: MacOSX port and multi-processor version. In *Proceedings of the first Sound and Music Computing conference - SMC04*, pages 177–183, 2004.

[12] Linux Audio Developers. Ladspa main website. `http://www.ladspa.org/`.

[13] M. A. Gerzon. Periphony: With-height sound re-production. *Journal of the Audio Engineering Society*, 21:2–10, January 1973.

[14] D.G. Malham and A. Myatt. 3-d sound spatialization using ambisonic techniques. *Computer Music Journal*, 19(4):58–70, 1995.

[15] A. Torger and A. Farina. Real-time partitioned convolution for ambiophonics surround sound. Institute of Electrical and Electronics Engineers, 2001.

[16] Nils Peters. Proposing spatdif - the spatial sound description interchange format. International Computer Music Conference, 2008.

[17] G. Potard and I. Burnett. Control and measurement of apparent sound source width and its applications to sonification and virtual auditory displays. Sydney, Australia, 2004. International Community for Auditory Display (ICAD), International Community for Auditory Display (ICAD).